Sistema para el diseño de las rutas diarias de distribución en el sector de alimentación

Henry Qiu Lo

Fecha de defensa: 23/10/2016
Directora: Elena Fernández Aréizaga
(Departamento de Estadística e Investigación Operativa)
Titulación: Ingeniería informática Especialidad: Computación
Centro: Facultad de Informática de Barcelona (FIB)
Universidad: Universidad Politécnica de Cataluña (UPC) - BarcelonaTech

Índice

1 Introducción

El proyecto trata sobre la creación de un sistema para el diseño de las rutas diarias de distribución en el sector de alimentación. La función principal del sistema será obtener las rutas de distribución que los vehículos de transporte tienen que seguir, de manera que minimizará los costes que pueden ser generados durante el proceso de distribución, y como consecuente, la empresa podrá obtener los máximos beneficios.

He escogido este trabajo porque para la creación de este sistema se requiere conocimiento en técnicas o algoritmos de computación, y gracias a los conocimientos que he adquirido durante la carrera podré poner a prueba mi consolidación sobre estos conocimientos.

A parte de los conocimientos de la especialidad, durante la realización del proyecto también podrá aplicar conocimientos adquirido desde asignaturas fuera de la especialidad. Además de los conocimientos adquiridos en la carrera, también podré investigar sobre técnicas desconocidas y aplicarlas.

En concreto, el sector de alimentación es para mí un sector muy interesante, porque durante el día a día podemos ver camiones de las grandes cadenas de supermercado repartiendo mercancía por la ciudad. A simple vista, parece algo muy cotidiano, pero en realidad hay un proceso complejo de planificación detrás. Las empresas siempre buscan maximizar sus beneficios y ocuparse de este estudio es la tarea principal de la investigación operativa.

En este trabajo comenzaré exponiendo las características, o información sobre el proceso de desarrollo de este trabajo, como el alcance y contextualización del proyecto, la planificación temporal, y la gestión económica y sostenibilidad. Posteriormente formalizaré el problema que trataremos, explicaré conceptos relacionados al problema, también presentaré los métodos que he utilizado para resolver el problema, como las herramientas que he usado para desarrollar todo el proyecto, por último, comentaré algunas posibles mejoras que se podría añadir y la conclusión que he obtenido al acabar este proyecto.

2 Alcance del proyecto y contextualización

2.1 Contexto

El diseño o planificación de las rutas de transporte para la distribución de productos es un problema muy serio para todas las grandes compañías en el sector de comercio, principalmente porque está estrictamente ligado con el coste (tanto en personal como en medios), para estas grandes empresas siempre es primordial reducir al mínimo los costes para conseguir el máximo beneficio en su actividad. Este problema es uno de los clásicos en los estudios de la investigación operativa.

En el sector de alimentación los distribuidores de los productos tienen que repartir la mercancía a las tiendas diariamente. Cada tienda tiene una cantidad de demanda diferente, y los recursos que dispone la compañía para realizar el transporte es limitado.

El proceso de distribución funciona de la siguiente manera: las tiendas de alimentación hacen sus pedidos antes de un cierto tiempo límite, la distribuidora o almacén central recoge estos pedidos, a partir de los recursos que dispone calcula las rutas que seguirán sus medios de transporte con el fin de minimizar los costes.

Una vez realizado esto en el momento del reparto los medios de transporte cargaran los productos de varias tiendas y repartirán en función de las rutas asignadas.

Para cadenas que tengan pocas tiendas la asignación de rutas de reparto no es difícil, las potenciales rutas de distribucion son limitadas y por lo tanto, no se necesita ningún sistema complejo que ayude en la determinación de rutas.

Sin embargo, para grandes cadenas de alimentación (como supermercados conocidos), donde un mismo almacén central puede ser proveedora de cientos de supermercados pequeños distribuidos en toda la ciudad, un sistema de asignación de rutas de distribución resulta imprescindible, ya que los almacenes suelen localizarse en lugares muy distantes respecto a las tiendas y hacer una mala distribucion de rutas puede suponer unos gastos enormes que son innecesarios.

Mi trabajo consiste en construir un sistema de asignación de rutas de distribución diarias de alimentación. El propio nombre indica que el uso este sistema estará destinado a las grandes empresas del sector de alimentación que necesitan diariamente realizar reparto de pedido de las tiendas.

Este sistema será usado diariamente porque los pedidos de las tiendas varía en cada pedido, además, cada cierto tiempo puede haber tiendas dado de alta, de baja, o que se hayan trasladado, como consecuencia asignar las mismas rutas cada día resulta inviable.

Para poder simular el comportamiento del proceso asignación de reparto, el sistema incluirá una interfaz para poder dar de alta los clientes (que en este caso son las tiendas), hacer pedidos de los productos, visualizar las rutas creadas, y dar información sobre el coste de las rutas calculadas. Este sistema no solamente simplificará las tareas del encargado de realizar la distribución de rutas en la compañía, sino que también beneficiará a los transportistas ya que cuando la mercancía que llevará y la distancia que recorrerá cada transportista será más equilibrada, y la carga de trabajo de cada transportista será más equilibrado. Así mismo, las tiendas que reciben la mercancía también se verán beneficiadas, ya que al estar adecuadamente asignadas las rutas el tiempo de espera medio de cada tienda en general se verá reducido.

El principal problema del diseño de la distribución de rutas es la dificultad en obtener la solución óptima para un problema complejo (en nuestro caso, cuando el número de tiendas a repartir es muy extensa). Es también objetivo del trabajo, resolver el problema utilizando dos métodos de resolución diferentes: programación lineal entera y métodos heurísticos basados en búsqueda local, posteriormente, comparar y evaluar las ventajas y desventajas de cada uno de los métodos analizando las características de estos algoritmos.

En resumen, el objetivo del trabajo es crear un sistema de diseño de distribución de rutas para ayudar a las grandes compañías a minimizar los costes, y valorar el comportamiento de diferentes algoritmos de resolución del problema de distribución de mercancía.

2.1.1 Actores (*stakeholders*)

2.1.1.1 Desarrollador

Es la persona encargada de hacer recerca, documentación e implementación de todos los requerimientos *software*. Será la persona responsable de escribir la memoria y toda la documentación necesaria. Es la persona que tiene que cumplir las fechas límites, bajo la supervisión de un tutor/a.

2.1.1.2 Directora del proyecto

Es la persona encargada de guiar, conducir y aconsejar al desarrollador. Son claves en la detección de errores en los proyectos. Como directora ha actuado Elena Fernández, profesora en el Departamento de Estadística e Investigación Operativa.

2.1.1.3 Beneficiarios

Cualquier compañía grande del sector alimenticio podría ser un beneficiario de nuestro proyecto, ya que nuestro producto está destinado al sector alimenticio para las empresas grandes.

2.2 Estado del arte

Al ser un problema muy relevante en el sector de distribución, era de esperar que en el mercado existiera una gran variedad de software que se encarga de gestionar la distribución del transporte de mercancías. Son programas de empresas que se dedican de manera especializada a la implementación de estos sistemas y como consecuente, son mucho más elaborados que un sistema que puede construir un alumno como trabajo de fin de grado.

Como ejemplo podemos encontrar Rutas,NovaTrans, que son software de cálculo de rutas de vehículos y flotas, ActiRuta, que es un software de optimización de rutas, Software ERP de Gestión de la Logística y Distribución de Ekon, etc.

Como características destacadas respecto a esa gran mayoría de programas software, el sistema que se diseñará en este trabajo es más específico para el sector de alimentación, está perfectamente adecuado a la gestión de una distribución de alimentos y al ser específico, la estructura del programa es muy simplificada, esto facilita el uso del programa para los encargados de realizar el diseño de rutas.

Otra ventaja de este proyecto respecto a los productos del mercado es que el coste de adquirir este sistema será mucho menor, al no disponer de un equipo formado para la implementación de un software la complejidad del programa es mucho menor, y como consecuencia, el coste para adquirir el producto debería ser mucho menor. Esto podría ser una ventaja para las empresas que necesitan este tipo de software pero no tienen intención de gastar mucho presupuesto, a pesar de que las funcionalidades del programa son limitadas, pueden ser suficiente para que estas compañías realicen el diseño de rutas.

Por contra, la mayoría de programas del mercado tienen un complejo sistema de gestión de logística, donde los usuarios pueden almacenar gran cantidad de información, también son capaces mostrar grandes detalles en los resultados obtenidos. El algoritmo que utilizan estos programas es desconocido, ya que no se ha encontrado ningún programa *"open source"*. Muchos de los programas son tan complejos que son aplicables en cualquier sector.

Como conclusión de este apartado, el planteamiento de este proyecto no tiene como finalidad dar una solución nueva al problema de diseño de distribución de rutas, la frontera de conocimiento del proyecto está en la aplicación de unos algoritmos ya conocidos, es muy probable que estos algoritmos hayan sido usados en los programas del mercado actual y que hayan sido perfeccionados. Para crear un producto más competente e innovador, se requeriría más recurso material, económico, personal, y tiempo.

2.3 El problema

Una vez tenemos el problema en la vida real definido, pasamos a formular el problema para poder ser resuelto con las herramientas que disponemos.

El problema del diseño de rutas de distribución es una extensión del problema del viajante de comercio(*TSP*). A diferencia del *TSP*, en nuestro problema tenemos restricciones de capacidad en los vehículos, lo cual obliga a distribuir la demanda entre los camiones y diseñar varias rutas en vez de una única ruta como en el *TSP*. Este tipo de problemas se denominan *"Vehicle Routing Problem" (VRP)*.[1]

El problema de enrutamiento de vehículos es uno de los problemas más conocidos de la optimización combinatoria, muy importante en la investigación operativa y en la computación.[2]

2.3.1 Objetivos

2.3.1.1 Implementación del sistema

Este problema también es conocido como uno de los problemas NP-completo, es decir, que no se ha encontrado ningún algoritmo y se desconoce la existencia de tal algoritmo que fuera capaz de resolver estos tipos de problemas en tiempo polinómico. Siendo este un problema NP-completo tan frecuentemente aplicado en la vida real, da mucho juego para hacer cierto estudio sobre este tema como un trabajo de fin de grado.

Concretamente, nuestro problema se aplica en el sector de alimentación. Un sistema completo de distribución de mercancías está formado por un cliente que realiza el pedido (en este caso los clientes son las tiendas que vendrán los productos al cliente directamente), y el almacén central que recoge los pedidos que realiza un cliente.

2.3.1.2 Análisis de diferentes méstodos de resolución

Este proyecto no solo se limita a formular el problema y a implementar dos métodos de resolución para el mismo. Además se ha realizado una experiencia computacional para evaluar el comportamiento de los algoritmos y evaluarlos empíricamente. Esto ha requerido, a su vez, la generación de datos numéricos para las intancias test.

Por ese motivo se ha creado una interfície que será usada por el almacén central. Esta interfaz le permitirá dar de alta nuevos clientes introduciendo información relevante de la tienda. También permitirá elegir los productos que cada cliente necesita para el próximo turno de distribución. Por lo tanto, para la implementación de esta parte, resulta imprescindible el uso de base de datos para guardar toda esta inforamción.

Dentro de esta interfaz, se le permitirá al usuario seleccionar el método de resolución del problema. En este punto, nuestro proyecto difiere a los otros productos existentes ya que en nuestro proyecto queremos ver el resultado de cada uno de los métodos, en cambio, en un producto acabado el usuario no tendría que tener conocimiento sobre ninguno de estos algoritmos ni lo necesita.

Por último, una vez calculada las rutas óptimas, para facilitar la visualización del usuario sobre las rutas obtenidas, se genera un mapa de grafos en el que los nodos representan localidades(una tienda o un almacén) y los arcos los caminos que deben coger los transportistas para llevar la mercancía a ese luegar.

En nuestro problema tenemos que definir ciertas variables que dependen de cada compañía en la vida real. Es el caso de las variables que representan el número de camiones, y la carga máximo de los camiones. En una compañía, se sabe cúales son las limitaciones que tiene, así que se sabe previamente el número de camiones de las que dispone y también la carga máxima que soporta cada camión.

Posteriormente encontramos las variables que dependen de los clientes que hay dados de alta, en este caso, las tiendas a las que el almacén central proveerá sus demandas. Estas variables representan la información detallada de los clientes, en concreto, el nombre de los clientes, la localización de los clientes que determinará la distancia que tendrá que recorrer los camiones, y la demanda de los clientes para saber cómo y cuánto peso distribuir en cada camión.

En resumen, en este trabajo se construirá una interfície que permitirá realizar operaciones de guardar, modificar, y eliminar en una base de datos, se resolverá un problema que consiste en un NP-hard problem mediante dos métodos diferentes, y se mostrará el resultado a través de una interfaz gráfica en forma de mapa de grafo donde se ilustrará el resultado. Finalmente, se hará análisis sobre el comportamiento de estos métodos escogidos.

2.4 Alcance / Obstáculos

Todo el proceso del diseño de las rutas de distribución en el sector alimenticio es difícil de representar. Hay muchos variantes de problemas de este tipo y estudiar todas las posibilidades es definitivamente inviable en este proyecto.

Para empezar, el sistema de pedidos en la vida real suelen hacerse diferente, en vez de ser el almacén central el que registra los pedidos, son los clientes los que realizan el pedido diariamente a través de un sistemas de pedido. Sin embargo, para nuestro proyecto donde importa el análisi de los métodos de resolución, resulta innecesario crear otra interfaz, en ese caso necesitaríamos crear un servidor que conectar diferentes clientes con el almacén y la carga del trabajo sería mucho mayor, así que queda fuera del alcance del proyecto crear un sistema de pedido para las tiendas al almacén.

Respecto al registro de la información de las tiendas, se ha minimizado la cantidad de información necesaria, ya que en nuestro caso no nos interesa ni el código de la empresa de la tienda, ni el teléfono, etc. Esto difiere a lo que suele ocurrir en programas complejos en los que se guarda gran cantidad de información de un cliente siempre que se da de alta.

En la definición del problema del diseño de rutas de distribución se ha ignorado un concepto muy relevante que es el concepto de tiempo. No haciendo referencia al tiempo de envío del camión sino el momento de reparto de cada cliente. La resolución del problema en este caso sería un poco diferente y más complejo. Para una futura implementación, estaría bién permitir a las tiendas introducir el tiempo ideal de reparto. Debido a la complejidad en este trabajo no se ha tenido encuenta.

También se podría haber planteado un problema de diseño de rutas con el fin de optimizar el número de camiones utilizados, es decir, suponiendo que la empresa contrata otra empresa de reparto, y el sistema permite obtener el número mínimo de camiones necesarios para obtener el mejor beneficio.

Otro de los potenciales planteamientos sobre el problema es que se puede asignar diferentes limitaciones a los camiones, cosa que podría ser muy habitual en la vida diaria. De la misma manera se podría asumir que el peso de los productos de demanda tuvieran cargas diferentes. Esto supondría otro tipo de problema NP-completo (Knapsack Problem) y la dificultad se dispararía.

Por último, el resultado se muestra en un mapa en el que las localizaciones de las tiendas están representadas en coordenadas respecto el almacén, efectivamente también se podría integrar el mapa de grafos con un mapa real de nuestra ciudad y guardar direcciones reales en la información.

En conclusión, definir el alcance del proyecto es muy importante ya que cuando intentamos simular un proceso de la vida real uno se da cuenta de la complejidad. Teniendo en cuenta que este es un proyecto de fin de grado, donde es un proyecto individual el tiempo no da para un sistema demasiado complejo. Así que el alcance de este proyecto no va más allá de crear un sistema de diseño de rutas de distribución donde se presupone un número limitado de camiones y una carga fija para la carga máxima que soporta los camiones y para el peso de los productos.

2.5 Metodología y rigor

Para el desarrollo software existen diferentes metodologías de trabajo, en este trabajo la metodología utilizada es la metodología ágil.

La metodología ágil consiste en un método iterativo, es decir en vez de hacer en línea el proceso de planteamiento, implementación, y prueba de golpe, separarlo en diferentes "Sprints" en el que dentro de cada Sprint se realiza planteamiento, implementación, y prueba para que, si se detecta algún error o se quiere añadir alguna funcionalidad, se pueda corregir ágilmente permitiendo así, mayor efectividad en el desarrollo.

Para guardar el código de la implementación del sistema se ha utilizado git. En concreto github que es el que más familiarizado debido a que se ha presentado en la universidad. Lo más útil para este proyecto es que permite visualizar todos los cambios hechos y revertir los cambios que no son correctos.

Para la planificación de tareas se ha utilizado Google Calendar, ya que es la herramienta a al que personalmente estoy acostumbrado a utilizar y esto me permite compaginar la planificación del trabjo con mi planificación de las actividades diarias. Cuando una actividad está acabada simplemente con dejar pasar el tiempo se marca automáticamente, de lo contrario cuando no me ha dado tiempo acabar alguna tarea lo arrastro al final de cada Sprint.

3 Planificación temporal

3.1 Descripción de las tareas/etapas

El proyecto se puede separar en dos partes principales: la implementación de los modelos de resolución y optimización del problema, y el desarrollo de la interfaz para la introducción o visualización de los datos y resultados.

La parte de los modelos de resolución y optimización del problema consiste en la resolución del problema: encontrar una distribución de rutas en la que se minimize el coste total del proceso de reparto. En nuestro trabajo usaremos dos disintos métodos de resolución: programación lineal entera y métodos heurísticos basados en búsqueda local, así que las etapas de esta parte consistirá en el planteamiento del problema para programación lineal entera y para búsqueda local, y la implementación de los algoritmos de resolución de programación lineal entera y de métodos heurísticos basados en búsqueda local.

La parte del desarrollo de la interfaz incluye crear la iterfaz del programa principal y la interfaz de muestra de los resultados en forma de mapa de grafo. Dentro de estas partes existen numerosas tareas: hay que definir las estructuras de datos y clases, dibujar los elementos de la interfaz, crear e integrar la base de datos en el programa, dibujar la solución, etc

A continuación se explican con más detalle las tareas implicadas en el orden que se ha planeado hacer:

3.1.1 Preparación del entorno

Para empezar el proyecto, primero se evalúa las herramientas necesarias para desarrollar todo el proyecto. Requiere un previo estudio y recerca sobre las herramientas disponibles que nos puede ser útiles, posteriormente, evaluar cuáles de ellas se adaptan más a nuestras necesidades. En concreto, consiste en descargar los programas, los complementos, las librerías, etc., todo lo que será necesario para poder arrancar el desarrollo del proyecto. Es claramente la primera etapa que deberíamos realizar para poder empezar a trabajar con un entorno estable. Además, es una tarea muy importante ya que si se quiere cambiar alguna decisión, por ejemplo, qué software utilizar o qué lenguaje utilizar con un proyecto a medias, puede suponer una carga bastante grande.

3.1.2 Programación lineal entera

Consiste en representar nuestro problema de asignación de rutas en un sistema de programación lineal entera, poteriormente, representar el problema de programación lineal entera en código de alto nivel, e implementar el algoritmo que resuelva el problema con determinadas entradas. Para realizar esta tarea es imprescindible tener la tarea de planteamiento acabado, ya que sin un buen planteamiento es muy difícil generar un buen código. La resolución de un problema de programación lineal entera no es muy muy rigurosa, ya que existe lenguajes o librearías que simplifican la implementación. Es una tarea que requiere conocimiento previo sobre programación lineal entera, en mi caso, es algo que lo he podido practicar en la asignatura de Investigación operativa. Sin embargo, en nuestro problema interviene el concepto de capacidad máxima de camión que es diferente a todos los problemas que he hecho en el curso. Esto me hace necesario consultar otras referencias como el libro **Vehicle Routing Problems, Methods, and Applications**.

3.1.3 Definir las estructuras de datos y clases

Incluye tareas como definir las estructuras de datos y clases, que consiste en determinar cómo estará estructurado el programa a nivel de código, a la vez analizar qué estructuras de datos se necesita crear para representar adecuadamente el problema. También incluye, dibujar los elementos de la interfaz, que consiste en plantear qué ventanas contendrá la interfaz, y qué elementos contendrá cada ventana. En concreto, esta tarea sirve principalmente para determinar cómo se activará las funcionalidades del programa para comprobar el cmoportamiento del sistema. Una vez planeada, hay que dibujarla con las herramientas que disponemos en función de la interfaz que hayamos escogido.

3.1.4 Integrar la base de datos en el programa

Incluye toda la tarea de definir y crear una base de datos para guardar la información necesaria y integrar esta base de datos en nuestro programa.

3.1.5 Búsqueda local

Consiste en escoger un algoritmo de métodos heurísticos basados en búsqueda local, representar nuestro probelma e aplicar el algoritmo escogido. Es una tarea que requiere cierto conocimiento sobre el algoritmo, y para ellos se ha consultado referencias en asignaturas de años anteriores como Inteligencia artificial. Posteriormente se necesita implementar una solución una vez el probelma está planeada para determinadas entradas.

Es una tarea que se podría haber hecho anteriormente, pero es mejor tener las estructuras de datos una vez definidas, el proceso es un poco más complejo ya que no utilizamos ninguna librería o API.

3.1.6 Dibujar elementos de la solución

Consiste en implementar la interfaz que muestra un mapa en forma de grafo donde se refleja la solución: nodos que representan tiendas y el almacén, y flechas que representan recorridos, indicando de esta manera la ruta que recorrerían los camiones.

3.1.7 Estudio final

Consiste en recoger los resultados, analizarlos y dar una solución. Incluye comprobaciones con casos generales, arreglo y revisión de código.

3.2 Estimación de tiempo

A continuación se muestra una tabla en el que aparece para cada tarea, una estimación del tiempo que se tardaría en realizarse.

Tarea	Duración(horas)	Dependencias
1) Preparación del entorno	20	-
2) Programación lineal entera	60	1)
3) Definir las estructuras de datos y clases	70	1)
4) Integrar la base de datos en el programa	70	1), 3)
5) Búsqueda local	90	1), 3), 4)
6) Dibujar elementos de la solución	90	1), 2), 5)
7) Estudio final	50	1), 2), 3), 4), 5), 6)
Total	450	

Tabla 1. Estimación de tiempo y dependencias de las tareas

El diagrama de Gantt se mostrará como Figura 1 en el apartado de anexos.

Hay tareas que se pueden realizar independientemente de otras, pero es necesario que algunas tareas estén acabadas para facilitar la implementación de otras. Es evidente que la primera tarea es la primordial para poder empezar el proyecto.

La segunda tarea se puede realizar completamente independiente de cualquier otra, de hecho el planteamiento suele estar hecho en papel antes de ser representado en código, algo que que debo elegir antes de tiempo bastante temprano, ya que determinará cómo se desarrollará el programa completo.

Para poder definir bien la base de datos estaría bién primero tener las estructuras de datos definidas, para integrar la base de datos en el programa sí que es inprescindible tener acabada la tarea de definir las estructuras, es una tarea que se puede hacer más tarde, pero siempre es recomendable tenerlo hecho al principio ya que ayuda a definir el esqueleto del sistema.

Las tareas de búsqueda local al ser implementado por mi ya que no existe ningún paquete, es la parte más extensa, y por ese motivo esta tarea dependería de la de definir estructuras de datos.

La tarea de integrar la base de datos también es bastante costosa ya que nunca he trabajado personalmente sobre ello, no requiere tareas previas.

Posteriormente, la tarea de dibujar elementos de la solución es evidente que sin tener el resultado obtenido no tiene ningún sentido, por eso esta tarea requiere las tareas de programación.

Por último, el estudio final solo se puede realizar cuando es la última activdad que queda, por lo tanto todas las otras tareas tienen que cumplirse antes de esta.

3.2.1 Valoración de alternativas y plan de acción

En el objetivo inicial del proyecto se iba a añadir un parámetro real que tener en cuenta: el tiempo. Es un parámetro importante en la vida real pero al introducirla en el TFG supondría una carga de faena mucho mayor. Por lo tanto, en este trabajo se ha eliminado el parámetro de "turno de reparto preferida" inicialmente propuesto.

La estimación de horas como su nombre indica, al fin y al cabo es una estimación. Durante el testeo de algunas tareas se ha encontrado problemas muy difíciles de resolver en los que era difícil encontrar el problema, hasta el momento, por ejemplo, la instalación de la API de AMPL en java no era fácil. También me he encontrado con pequeños errores de concepto en el planteamiento de los problemas. Al ser tareas muy importantes se ha tenido que dedicar más tiempo a esta parte que las partes de implementar o dibujar. Como alternativa, esto no supone un problema muy grande ya que cuando la implementación finalmente es la correcta, podemos aconseguir pequeñas ventajas en la implementación.

La dificultad principal del trabajo para mí personalmente es la parte de búsqueda local, y por ese motivo como alternativa para afrontarlo en la planificación le he asignado más horas, por lo que tendré más tiempo de investigación y entendimiento.

También resulta frecuente un atraso del proceso en el tiempo por motivo de pequeños errores, la alternativa final es simplemente dedicarle más horas y sacrificar tiempo personal en el día día. Se puede concluir que con todo el tiempo que dispongo (un cuadrimestre más tres meses) antes de la presentación podré superar los problemas y acabar el proyecto a tiempo.

3.2.2 Especificación recursos utilizados

Recursos humanos usados:
Desarrollador *software*: será el único personal que se encargará de todo el proyecto debido a que no se trata de un proyecto con mucha complejidad.

Recursos materiales usados:
Ordenador portátil: será el único material necesario para desarrollar el proyecto.

4 Gestión económica y sostenibilidad

4.1 Autoevaluación del dominio actual de la competencia de sostenibilidad

Hoy en día cada vez se tiene más consideración en las consecuencias que pueden generar un proyecto en la sostenibilidad. Sé analizar perfectamente la sostenibilidad desde el punto de vista medioambiental, social y económica de las TIC: durante el proceso de planteamiento de un proyecto, ya evaluamos el proyecto qué impacto social y económico tendrá, y dependiendo de este impacto evaluamos la viabilidad de nuestro proyecto, respecto a la dimensión medioambiental, es más dícil de calcular, pero es fácil estimar que el impacto de nuestros proyectos no deberían tener impactos medioambiental graves.

Conozco bién los conceptos de creatividad e innovación y comprendo las técnicas, sin embargo, a veces no soy capaz de aportar nuevas ideas y soluciones en un proyecto tecnoógico para hacerlo más sostenible, normalmente mis ideas se centran en mejorar aspectos más técnicos que no los que afecten a la sostenibilidad.

Por otro lado, he podido observar toda la problemática y consecuencias asociada a la seguridad, justicia social a través de lecturas diversas, y es por ese motivo que hoy en día se tiene tanto en cuenta la sostenibilidad.

Asímismo, soy capaz de valorar fácilmente si un proyecto TIC contribuye a mejorar el bién común de la sociedad, de hecho, mejorar el bién común de la sociedad era el objetivo principal de la aparición de la tecnología. No soy capaz de incluir indicadores para estimar cómo los proyectos contribuyen a mejorar el bien común de la sociedad, sin embargo, siempre trato de maximizar el impacto positiva de mi actividad profesional, ya que poder ver que un proyecto mío pueda tener impacto positivo en la sociedad despierta en mi mucha sensación de satisfacción.

Es cierto que incomprendo complatamente las partes económicas de un proyecto, ya que nunca me ha desperatdo interés sobre el tema del análisis económico.

Por la otra banda, sé valorar perfectamente las implicaciones del trabjo colaborativo en un proyecto TIC, ya que desde el inicio de la carrera me han inculcado los valores de la colaboración y trabajo en equipo para poder realizar un buen trabajo colectivo.

Finalmente, desconozco completamente los principios deontológicos relacionados con la sostenibilidad de un proyecto TIC. Como conclusión, puedo decir que aún necesito profundizar mi conocimiento en el área de la sostenibilidad ya que hoy en día está cogiendo cada vez más importancia.

4.2 Dimensión económica de la matriz de sostenibilidad: presupuesto

4.2.1 Costes Directos

Puesto	**Salario (€/hora)**	**Total (€)**
Desarrollador *Software*	16	16 * 450 = 7200

Tabla 2. Coste de personal.(El salario de un desarrollador viene dada por la referencia: 5)

4.2.2 Costes materiales

Objeto	**Valor (€)**	**Valor después de amortización (€)**
Ordenador portátil	600	38.35

Tabla 3. Coste en *Hardware*

El cálculo de la amortización se ha hecho de la siguiente manera:

$$\frac{Preciodecompra(600) * Horasdeuso(450)}{Vidaútil(4) * díaslaboralesaño(220) * Horastrabajodía(8)}$$

Donde se supone que el ordenador tiene menos de 4 años.

Software	**Precio (€)**
Google Chrome	0
Eclipse Neon	0
Scen Builder	0
AMPL	0
Adobe Acrobat Reader	0
Team Gantt	0
SQLite Manager	0

Tabla 4. Coste en *Software*

El coste de los programas sale completamente gratuito.

4.2.3 Costes indirectos

Objeto de uso (450 h)	**Precio (€)**
Luz	30
Agua	20
Material de oficina	10
Fibra óptica	240
Transporte	200
Total:	700

Tabla 5. Costes indirectos

4.2.4 Costes inesperados

Posición	**Horas**	€/hora	Total (€)
Desarrollador *Software*	20	16	20 * 16 = 320

Tabla 6. Costes inesperados

En nuestro caso se puede dar cuando el planteamiento o la estimación de tiempo de las tareas es incorrecta y se producen retrasos, como consecuencia habrá que pagarle horas extras.

4.2.5 Costes totales

Aparece el concepto de contingencia, en el que interviene un parámetro que indica la probabilidad de riesgo. Se hace una suposición de que será del 5%.

Concepto	**Coste estimado (€)**
Directos	7200
Material	38.35
Indirectos	700
Inesperados	320
Subtotal	**8258.35**
Contingencia (5%)	412.9175
Total	**8671.27**

Tabla 7. Costes totales

Como mecanismo de control de desviación podemos omitir algunas tareas específicas como por ejemplo tareas de testeo que podrían ser innecesarios cuando el programa desarrollado no tiene muchos errores, sin embargo, puede dar resultado a un mal producto elaborado. Como mejor prevención sería comprar un ordenador a precio más reducido, ya que para el desarrollo de este proyecto no se requiere un ordenador muy avanzado.

4.3 Matriz de sostenibilidad

A continuación se muestra la matriz de sostenibilidad:

	Económico	Ambiental	Social
PPP	8/10	3/10	10/10

Tabla 8. Matriz de sostenibilidad

En la dimensión económica la estimación de coste económico del proyecto tengo duda sobre el tiempo real necesario del proyecto, ya que al final la estimación puede desviarse mucho de la realidad. El salario estimado de los recursos humanos tampoco es preciso, ya que puede variar mucho en función de los empleados contractados. Los problemas de costes que se quiere abordar se pueden resolver pidiendo prestaciones al banca para invertir en proyectos. El coste del programa creado mejor sustancialmente respecto a los existentes, ya que es un proyecto relativamente más pequeño y el coste que hay detrás del proyecto es muhco más pequeño que el coste de los productes en el mercado, sin embargo, lógicamente el beneficio también será menor.

En la dimensión ambiental, tengo muy poco conocimiento sobre el impacto ambiental, tampoco me he planeado alguna metodología para mejorar el impacto ambiental, ya que el impacto ambiental que está generando el proyecto ya es muy pequeño. Respecto a otros productos este producto mejorará en el impacto ambiental ya que los recursos que necesita gastar es mucho menor.

En la dimensión social, debido a que en el mercado ya existe gran cantidad de productos relacionados, el valor útil que aport mi proyecto no es muy grande, sin embargo, a nivel personal me ha ayudado a desarrollar mis capacidades de planificación y síntesi.

5 Definición y planteamiento del problema

Cuando pretendemos resolver un problema de la vida real con un sistema informático necesitamos hacer una abstracción del problema. Es decir, tenemos que definir los elementos relevantes que afectarán a la resolución del problema, elegir una forma de representación más adecuada, y a partir de aquí, desarrollar el proceso de resolución.

Para empezar, definiremos los elementos o conceptos claves del problema:

- Almacén central: es el lugar donde procesa los pedidos de las tiendas distribuidoras y carga los productos en los vehículos de reparto, generalmente camiones en el sector de alimentación. La importancia de este dato en nuestra resolución del problema recae en que es el inicio y final de la ruta de los vehículos.

- Tiendas distribuidoras: son los que venden los productos de cara al público. En nuestro sistema estas tiendas tienen el rol de cliente, ya que son los que realizan el pedido diariamente, el almacén central es el que servirá a estas tiendas.

- Camiones: en concreto, para la resolución del problema nos interesa el número de camiones que dispone la empresa y la carga máxima que soporta cada camión. Para abreviar, estamos suponiendo que el vehículo que utilizarán las empresas para la distribución de la mercancía será camiones.

- Pedido: la mercancía que será distribuida por el almacén para los clientes. En nuestro sistema únicamente tiene relevancia el peso de los productos que pertenecen a un pedido y no en qué productos consisten estos pedidos, ya que solamente necesitamos saber si todos los productos de una tienda caben un determinado camión

-Coste: parámetro que utilizamos para medir el coste de transporte de los camiones. En nuestro caso el coste puede ser la distancia entre una tienda y otra. En este problema el coste influye en en función de qué parámetro se determinará que una ruta es mejor que otra.

Dentro del sistema, tanto el almacén como las tiendas distribuidoras tienen una localización concreta, que serán representados en forma de coordenadas, cada cliente tendrá además, un nombre que lo identifica, además, se guarda el último pedido realizado por cada cliente para facilitar el proceso de pedido, ya que normalmente la mercancía que pide una tienda puede varía poco de un día a otro.

El problema se puede representar en forma de grafo dirigido, donde existe un nodo que representa el almacén central, y n nodos, con $n = |V|$ que representan todas las tiendas distribuidoras. Los caminos que los camiones pueden seguir vienen representadas por las aristas, cuando una arista va del nodo i al nodo j, quiere decir que existe un camino por estas tiendas saliendo desde i para llegar a j. Todos los camiones saldrán desde el almacén, así que todas las rutas tienen como punto de partida el nodo almacén, y acabarán en el almacén una vez haya vaciado el camión.

Explicado más formalmente: tenemos un grafo G=(V,E), donde V son los vértices y E las aristas, que asociada a cada arista e=(i,j) hay un coste, que se denota por $C_{i,j}$. Asociado a cada vértice $i \in V$ hay una demanda d_i, además, tenemos una flota de K vehículos homogéneos, todos con la misma capacidad q.

En este proyecto se hace los siguientes hiótesis:

- Sólamente existe un almacén, ya que diferentes almacenes se encargará de tiendas diferentes.

- Todos los productos en un pedido será encargado por un sólo camión, es decir los productos pedidos por una tienda no pueden estar distribuido en dos camiones, esto implica que los por cada tienda pasará un sólo camión

- Cuando un camión vuelve al almacén ya no hace más reparto durante el día

- Los productos se representan en cajas del mismo peso, esta asunción se realiza porque si los productos que pueden caber en un camión tienen pesos diferentes, surgiría otro problema a tener en cuenta en la optimización, que es el "Knapsack problem".

- En el sistema no aparece el concepto de tiempo, se presupone que en un día todos los clientes podrán ser servidos, ya que tampoco no sería responsabilidad del sistema si no existen recursos suficientes para satisfacer la demanda de todos los clientes en un día. Además, esto implicaría de nuevo aumentar la complejidad del problema ya que se introduciría ventanas de tiempo.

5.1 Definición de variables

Como se había explicado anteriormente, el problema está representado en un grafo, donde los nodos están representados por los clientes y el almacén, y las aristas representan los caminos. Para saber cúales son las rutas que deberán seguir los camiones para minimizar el coste necesitaremos guardar las rutas, y unas rutas no son más que nodos por el que un camión tiene que pasar. Así que una variable que nos permita saber por qué rutas pasarán los camiones será necesaria. Para esto se utilizará la variable *Xij*, donde i y j tendrán valores comprendidos entre 0 y el número de nodos. Será una variable binaria, es decir, solamente podrá tener dos valores: 0 o 1. Cuando la variable coge valor 1 querrá decir que hay un camión que pasa desde el nodo i hasta el nodo j.

Durante el proceso de transporte cada camión llevará una determinada carga específica dentro de su límite de capacidad. Como cada tienda tiene una demanda de carga diferente, a medida que van pasando por las tiendas van depositando la mercancía pedida por las tiendas, esto hace que la carga de los camiones en diferentes puntos del mapa varíe. Para reflejar esta información se ha utilizado la variable U_i, donde i tendrá valores comprendidos entre 0 y el número de nodos, y U_i tendrá como valor la cantidad de producto depositado por un camión después de pasar por el cliente i. El uso de esta variable juega un papel esencial que se explicará en el siguiente apartado.

La función objetivo que se quiere minimizar es el coste total del proceso de reparto. El coste de todo el proceso de reparto será la suma del coste que generará cada camión en su reparto. En este problema, como habíamos supuesto que el número de camiones será limitado, el coste será principalmente el combustible que se gasta durante el proceso de reparto. Podemos asociar el coste del combustible con la distancia recorrida, al ser el precio del combustible una constante para todos los camiones, el coste de nuestro problema será simplemente, la distancia recorrida.

A parte de las variables, también existen otros parámetros que describen el problema:

- El parámetro d_i contiene: para cada nodo i dentro del número de vértices, la demanda de los clientes, en concreto, el almacén que es el nodo 0 tiene demanda 0.

- El parámetro k indica el número de camiones disponibles para realizar el reparto.

- El parámetro q indica la carga máxima de los camiones. Como habíamos mencionado anteriormente, en el proyecto asumimos que la carga máxima que soporta todos los camiones es la misma.

- El parámetro m es un número suficientemente grande para que la restricción de MTZ pueda funcionar como se ha descrito en el apartado siguiente.

5.2 NP

NP (*nondeterministic polynomial time*), es el acrónimo que se utiliza en la teoría de la complejidad computacional para referirse al conjunto de problemas que se puede resolver en tiempo polinómico por una máquina de Turing no determinista. Dado una solución los problemas NP se pueden comprobar en tiempo polinómico.

También tenemos los problemas de complejidad P, que son los que se pueden obtener una solución en tiempo polinómico. Está claro que los problemas P son unos subconjuntos de los problemas NP, ya que si podemos obtener una solución en tiempo polinómico, podemos comprobar con el resultado en tiempo polinómico.

Actualmente existe un enorme debate, entre si NP también estaría incluido en P, es decir, si podemos comprobar P = NP, es decir, si pudiéndose comprobar que los problemas en tiempo polinómico se puede resolver en tiempo polinómico. De hecho, el dilema ¿P = NP ? es uno de los problemas del milenio, esto refleja su importancia.

5.2.1 NP-completo

Las clases de problemas con complejidad NP-completo son los subconjuntos de los problemas NP, tal que cualquier problema en NP se puede reducir a cada uno de los problemas de NP-completo. Son los problemas más "difíciles" de NP. Si se puede llegar a demostrar que un problema NP-completo se puede resolver en tiempo polinómico, se podrán resolver todos los otros problemas en tiempo polinómico.

Actualmente no se ha encontrado aún ningún algoritmo capaz de resolver estos problemas en tiempo polinómico. Pero tampoco se ha demostrado que no pueda existir ningún algoritmo que los pueda resolver en tiempo polinómico.

Estos son los problemas NP-completo más famosos:

- Problema de la clique
- Problema de la mochila (knapsack)
- Problema del ciclo hamiltoniano
- Problema de la clique
- Problema del viajante

6 Métodos de resolución del problema

6.1 Programación lineal entera

Los problemas de programación se caracterizan por tener una función objetivo lineal que optimizar, ya sea maximizando o minimizando. Esta función viene sujeta por una serie de restricciones representadas como relaciones lineales. Resolver el problema significa encontrar valores para las variables de la función objetivo, de manera que el función objetivo fuese el óptimo (máximo o mínimo) cumpliendo todas las restricciones.

Podemos plantear nuestro problema como un problema de programación lineal, con restricciones adicionales de integridad sobre las variables. Tenemos un obetivo a minimizar: el coste total del reparto de mercancía. También tenemos una serie de restricciones a complir: por ejemplo, la capacidad del camión está limitada, el número de camiones disponibles también está limitado, etc. En concreto, debido a que las variables que elejiremos tendrán que ser números enteros, se dice que el problema es de programación lineal entera.

La resolución manual de los problemas de programación lineal es compleja y conlleva mucho tiempo, para ello se ha inventado optimizadores o *Solvers* como el *CPLEX* que ejecutan los algoritmos de resolución de problemas de programación lineal. Gracias a esta tecnología solamente tenemos que plantear el problema correctamente para que los optimizadores nos devuelvan el resutlado en caso de que haya.

Para poder hacer un buen planteamiento, lo que se necesita primero es decidir cuáles serán o qué representarán las variables del problema.

6.1.1 Formulación

$$min \sum_{i \in n} \sum_{j \in n} C_{ij} X_{ij}$$

$$\sum_{j \in V} X_{ij} = 1 \quad \forall j \in V\{0\}, \qquad (1)$$

$$\sum_{j \in V} X_{ij} = 1 \quad \forall i \in V \setminus \{0\}, \qquad (2)$$

$$\sum_{i \in V} X_{i0} <= K, \qquad (3)$$

$$\sum_{j \in V} X_{0j} <= K, \qquad (4)$$

$$u_i \geq u_j + d_j - M(1 - X_{ij}) \qquad (5)$$
$$u_0 = 0 \qquad (6)$$
$$d_i \leq u_i \leq q \qquad (7)$$
$$X_{ij} \in \{0, 1\} \quad \forall i, j \in V.$$

En la primera restricción se garantiza que todos los nodos clientes tienen una flecha de entrada, es decir, cada cliente será servido.

En la segunda restricción se garantiza que desde cada nodo cliente solo salga una flecha, es decir, complementandose a la restricción anterior, se garantiza que por cada cliente sólo puede pasar un camión.

La tercera restricción exige que el número de flechas que llega al nodo 0 tiene que ser k, es decir, que número de camiones que vuelvan al almacén sea k.

La cuarta restricción exige que el número de flechas que sale del nodo 0 tiene que ser k, es decir, complementando a la restricción anterior, el número de camiones que salen desde el almacén tiene que ser igual al número que vuelven.

Las restricciones (5), (6) y (7) constituyen las restricciones de MTZ.

6.1.2 *Miller-Tucker-Zemlin*

Las restricciones de *Miller-Tucker-Zemlin*(MTZ) son unas restricciones de eliminación de subcircuito, han sido y siguen siendo de uso regular para modelar una variedad de problemas de enrutamiento.

En la formulación del problema de enrutamiento la mayor dificultad es evitar la formación de subrutas. Es decir, el caso en que un camión sale desde una tienda y acaba en la misma tienda, haciendo un ciclo sin haber salido desde el amacén.

Para evitar este problema añade una restricción en la que, la carga del vehículo tiene que ser descendiente a medida que pasa por las tiendas, hasta llegar a 0 al cuando vuelva al almacén. Esto evita que se formen subcircuitos, porque en un ciclo cerrado la carga del vehículo no podrá ser descendiente cuando vuelva al punto inicial.

$$u_i \geq u_j + d_j - M(1 - X_{ij})$$

$$u_0 = 0$$

$$d_i \leq u_i \leq q$$

La formulación de la restricción se puede adaptar a diferentes contextos, en nuestro problema, Ui representa la cantidad de producto depositada por el vehículo después de visitar i, fijando a 0 U0 para indicar que los camiones tienen que volver vacíos al almacén. Además, cada Ui tiene que estar entre la demanda del cliente i y el límite que soporta el camión. Además, esta formulación tiene una peculiaridad, que es que solo tendría sentido para una arista en la que pasa algún camión, por ese motivo se ha añadido en la restricción un parámetro de control, en la M es un valor muy grande, y cuando Xij es 0, es decir, no pasa ningún camión, la parte derecha de la restricción será negativa, y siempre se cumplirá, de esta manera esta restricción no afectaría.

Existe otra manera de evitar el problema de formación de subcircuitos, de hecho, es más intuitiva que la propuesta en MTZ. Consiste en limitar el número de subgrafos creados.

$$\sum_{I \in S, j \in S} X_{i,j} \leq |S| - 1 (S \subsetneq V, |S| > 1)$$

Entonces, ¿por qué motivo no usamos esta restricción más entendedora? La respuesta es el coste algorítmico: el número de restricciones que se deben imponer es exponencial, el problema se convierte inmensamente grande y resulta inviable resolverlo. Por este motivo, las MTZ son las más utilizadas para resolver problemas de enrutamiento hoy en día.

6.2 Métodos heurísticos basados en búsqueda local

Los métodos heurísticos suponen la existencia de una función de evaluación que mide la distancia estimada a un objetivo. Esta función de evaluación se utiliza para guiar el proceso haciendo en que cada momento se seleccione el estado o las operaciones más prometedores.

A veces el camino para llegar a la solución no nos importa, y por ello buscamos el óptimo en el espacio de soluciones. Queremos la mejor de entre las soluciones posibles alcanzable en un tiempo razonable. Tenemos una función que nos evalúa la calidad de la solución, pero que no esta ligada a ningún coste necesariamente. Los operadores nos mueven entre soluciones vecinas existentes.

Entre los algoritmos que utilizan búsqueda local más empleados podemos mencionar *Hill climbing y simulated annealing*.

6.2.1 *Simulated Annealing*

Es un algoritmo de búsqueda heurística, en concreto, el objetivo general de este tipo de algoritmos es encontrar una buena aproximación al valor óptimo de una función en un espacio de búsqueda grande. A este valor óptimo se lo denomina "óptimo global". Es un tipo de Hill-Climbing estocástico, es decir, elegimos un sucesor de entre todos los posibles según una distribución de probabilidad. El algoritmo consiste en hacer paseos aleatorios por el espacio de soluciones. por lo tanto, elsucesor podría ser peor.

El algoritmo se ha insipirado en el proceso físico de enfriamiento controlado (Cistalización, templado de metales). Se calienta un metal/disolución a alta temperatura y se enfría progresivamente de manera controlada. Si el enfriamiento es adecuado se obtiene la estructura de menor energía (mínimo global).

Los problemas del *Simulated annealing* son adecuados problemas grandes en los que el óptimo está rodeado de muchos óptimos locales,indicado para problemas en los que encontrar una heurística discriminante es difícil (una elección aleatoria es tan buena como otra cualquiera). Unas de las posibles aplicaciones es la resolución de problemas TSP.

En el algoritmo de *Simulated annealing* se distingue dos partes principales, la fase cosntructiva y la fase exploratoria.

6.2.2 Fase constructiva

En la fase constructiva se construye una solución inicial válida. Esta solución puede no ser muy buena, pero tendrá una determinación muy importante de cómo irá encaminado la búsqueda del óptimo.

Estos son los parámetros que necesitan los problemas: temperatura, parámetro de control que ayudan en la decisión del nuevo resultado. Energía, calidad de la solución. Función de aceptación, permite decidir si escoger un nodo sucesor.

- Clark & Wright

Es uno de los algoritmos heurísticos más famosos para la planificación de rutas. Se aplica a problemas donde el número de vehículos no está fijado. Su teoría se basa en que dos rutas de la forma "0-...-i-...-0" y "0-...-j-...-0" se pueden unir de la forma "0-...-i,j-...-0", de manera que se produce un ahorro en concepto de distancia por no tener que volver al nodo inicial entre i y j.

El procedimiento del algoritmo consiste en el siguiente:

- Para empezar se construye una matriz de ahorros. Consiste en una matriz M de n * n, donde N es el número de clientes. Para cada Mij se guarda el ahorro entre los clientes i y j. El ahorro se calcula de la siguiente manera: $M_{i,j} = C_{i,0} + C_{j,0} - C_{i,j}$, donde i y j representan el nodo cliente y 0 representa el almacén. El concepto de ahorro, como su nombre indica, significa el coste que no tenemos que gastar si vamos desde i a j directamente en vez de pasar por el origen, cuando tenemos un valor de 0 en alguna posición (excepto en la diagonal que el valor no es significante) significa que no ahorramos nada en ir desde i a j directamente. Otras de las características de la matriz es que es simétrica, eso es porque M[i][j] = M[j][i].

- El siguiente paso consiste en ordenar los pares i,j en orden decreciente en concepto de ahorros. Como la matriz es simétrica, con tener en cuenta la mitad de la parte inferior de la diagonal de la matriz es suficiente.

- Se recorre los pares ordenados, para cada par ordenado se verifica si al añadirla como resultado se está cumpliendo las restricciones. Las restricciones son parecidas al método anterior: desde cada nodo sólo puede salir una arista, a cada nodo sólo puede llegar una arista, la suma de la carga total de las subrutas formadas tiene que ser menor que la capacidad del camión. Si al añadir el par no viola ninguna restricción, se añade a la solución, en caso contrario se rechaza el par y continúa con el siguiente. Cuando se recorren todos los pares, en la solución quedan las subrutas creadas que sabemos que satisfacen las restricciones, a cada nodo que no recibe ninguna arista se le añade la arista desde el almacén, y cada nodo que no llega a ninguna arista se añade una arista hacia el almacén.

Podemos observar que en nuestro algoritmo no se comprueba la restricción en que el número de rutas creadas tiene que ser menor que el número de almacén, esto es debido a que es posible que primero encontremos una solución inicial que podrá ser mejorada, y con la mejora en la siguiente fase se logre obtener una solución mejor, cumpliendo la restricción del número de camiones limitados.

Un hecho relevante es que, mientras no exista una demanda de un cliente mayor a la capacidad del camión, el método de *Clark & Wright* siempre obtiene solución. Esto es debido a que en nuestro caso no estamos teniendo en cuenta la limitación del número de camiones, en el caso peor tendremos tantas rutas como clientes, es decir, se utilizará tantos camiones como clientes hayan. Obviamente, en nuestro programa posteriormente se comprobará que si no aconseguimos una solución con menos rutas que el número de camiones devolverá como resultado problema no factible.

6.2.3 Fase de exploración

En la fase de exploración se exploran diferentes resultados alternativos al resultado inicial y se aceptan resultados nuevos dependiendo de unos parámetros de probabilidad. En comparación con otros métodos, no solamente acepta resultados mejores, sino que acepta con cierta probabilidad soluciones peores, esto es porque muchas veces para encontrar el óptimo es necesario pasar por algunas soluciones malas, y pasando por solucionas malas nos permite a veces alcanzar a soluciones mejores que las anteriores.

En esta fase se determinan las operaciones que se realizarán para la actualización. En este trabajo las operaciones que se han escogido son: intercambiar dos clientes asignadas a diferentes rutas, reasignar un cliente de ruta.

En el primer tipo de operación se intercambia un cliente asignado en una ruta de reparto por otro cliente asignado en otra ruta. La desventaja de esta operación es que el número de rutas y el número de clientes visitados en una ruta no varía, esto hace que si en la solución inicial se necesitan más camiones de lo necesario, no se podrá obtener nunca una solución con menos camiones aunque éste exista.

Para solventar este problema se hace la segunda operación, donde un cliente asignado a una ruta será reasignada a otra ruta, de esta manera, es posible que todos los clientes que originalmente formaban parte de una ruta pasarán a formar parte de otras rutas de manera distribuida, y así será posible alcanzar a soluciones significativamente mejores que la inicial.

Para ambas operaciones se requieren revisar si se siguen cumpliendo las restricciones de carga máxima de los camiones, en caso de que se cumpla se procede a analizar si merece la pena aceptar la nueva solución, en caso contrario esta solución se rechazará.

6.2.4 Formulación

Primero se realiza la fase constructiva, como se ha explicado anteriormente, primero se construye la matriz de ahorros, a partir de esta se guardan estos ahorros en una estructura de datos de ordenación, para obtener los pare de clientes y los ahorros correspondientes. Al final de la primera fase devuelve una estructura que contiene el recorrido de cada una de las rutas creadas.

Posteriormente se realiza la fase de exploración en base a las rutas creadas. La exploración se hace de la siguiente manera: se realizan operaciones de intercambio de clientes, manteniendo las rutas existentes, se comprueba si la nuva solución creada es una solución factible, si la es, se evalúa si se acepta la nueva solución, si se acepta, se actualiza el mejor resultado obtenido hasta el momento. Este proceso se itera hasta que la temperatura se ”relaje” y no podamos seguir mejorando, a partir de aquí, se intenta mejorar la solución realizando otra operación: cambiar cliente de ruta, y de nuevo, se realiza repite el bucle anterior. El algoritmo acaba cuando se acaba este doble bucle de exploración de resultados.

6.3 Comparación de los métodos

Los dos métodos sirven para resolver el problema de enrutamiento de vehículos, pero las características de estos métodos los hace muy diferentes: - Los métodos de programación lineal entera nos garantizan siempre la solución óptima en caso de que exista, en cambio, los métodos heurísticos no siempre se garantizan encontrar la solución óptima, principalmente porque en los segundos interviene el concepto de probabilidad.

- Los algoritmos de los métodos heurísticos no pueden hacer una exploración sistemática de los problemas, en cambio, los métodos de programación lineal sí lo hacen, aquí está el motivo por el que un método garantiza encontrar el óptmo y otro no. En los métodos heurísticos la función heurística se usará para podar el espacio de búsqueda (soluciones que no merece la pena explorar).

- El coste temporal de los métodos heurísticos será mucho menor, en cambio, la programación lineal busca la resolución exacta de un problema NP-completo, por lo tanto tiene un coste temporal exponencial en el pero caso.

- El coste de implementación de los métodos de programación lineal es mucho menor teniendo en cuenta que existen optimizadores, ya que podemos formular el problema y ejecutar el programa de optimización para que haga todo el trabajo de resolución, en cambio, la programación de los algoritmos de los métodos heurísticos tiene una complejidad significante. De los métodos heurísticos es difícil aplicar librerías previamente implementadas porque para cada tipo de problema, la heurística que se utiliza puede variar.

Entre las diferencias mencionadas anteriormente, el punto que más marca la diferencia es el tiempo de ejecución de sus respectivos algoritmos.

Cuando se analiza la complejidad temporal de un problema se toma como referencia el coste temporal en el peor caso. Actualmente el coste temporal de un problema es el coste temporal del mejor algoritmo que lo resuelve.

Para los problemas de programación lineal entera, el coste temporal es exponencial. Siendo un problema NP-completo actualmente es difícil mejorar el coste. Actualmente unos de los método de resolución del programación lineal entera más utilizados es el *Branch and Bound* (Ramificación y poda). Esta técnica se suele interpretar como un árbol de soluciones, donde cada rama nos lleva a una posible solución posterior a la actual. La característica de esta técnica es que el algoritmo se encarga de detectar en qué ramificación las soluciones dadas ya no están siendo óptimas, para "podar" esa rama del árbol y no continuar malgastando recursos y procesos en casos que se alejan de la solución óptima. Sin embargo, el caso peor el coste del algoritmo sigue siendo exponencial.

El coste temporal del *Simulated annealing* está acotado por el número de iteraciones que se harán, y este valor depende de la temperatura y el parámetro de relajación. Estos parámetros son de libre elección, en nuestro caso, hemos escogido dos tipos de operaciones para realizar la mejora, además, escogemos los mismos valores de temperatura y parámetro de relajación para las dos operaciones.El coste del algoritmo de *Simulated annealing* que hemos implementado es $r^2 * T^2$, donde T es la temperatura y r el parámetro de relajación.

A continuación se presenta un gráfico comparativo para de los tiempos de ejecución con los dos métodos de resolución diferentes.

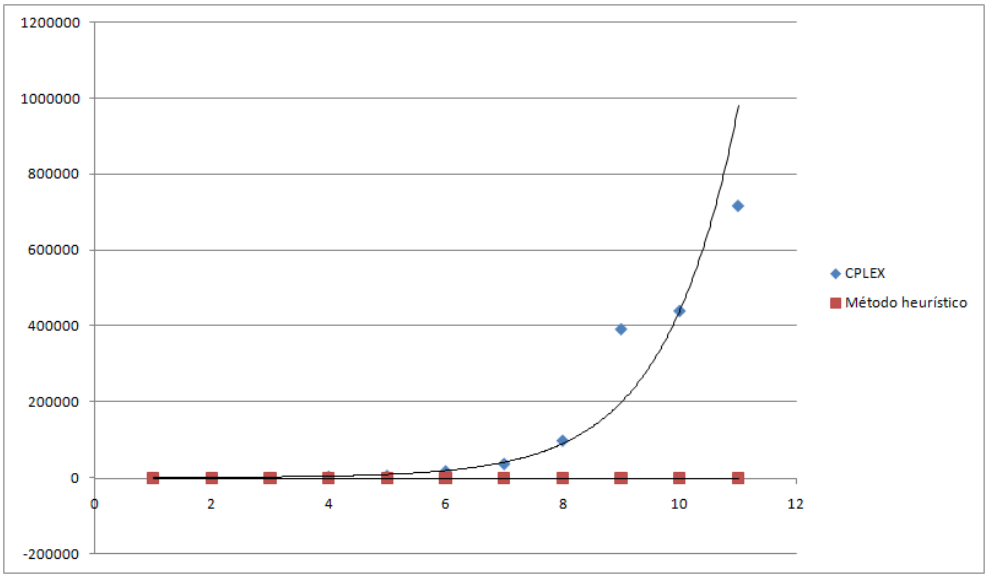

Figure 1: Gráfico de tiempos de ejecución.

CPLEX(ms)	Método heurístico(ms)	Número de clientes	Número de camiones	Capacidad de Camiones
576	5	7	2	20
710	6	8	2	30
1052	6	9	2	40
3812	7	10	2	40
6032	8	11	2	40
18008	8	12	2	40
37581	10	13	2	50
98816	18	14	2	50
392708	40	15	2	50
440733	43	16	2	50
717669	62	17	2	50

Figure 2: Tabla de tiempos de ejecución.

La prueba consiste en incrementar el número de clientes, ejecutar el programa con los dos métodos de resolución y observar la variación del tiempo de ejecución para ver cómo afecta el número de clientes a la dificultad de resolución del problema.

En las dos primeras columnas se encuentran los tiempos de ejecución en milisegundos de CPLEX y Método heurístico. Siguientemente podemos encontrar parámetros como el número de clientes, el número de camiones

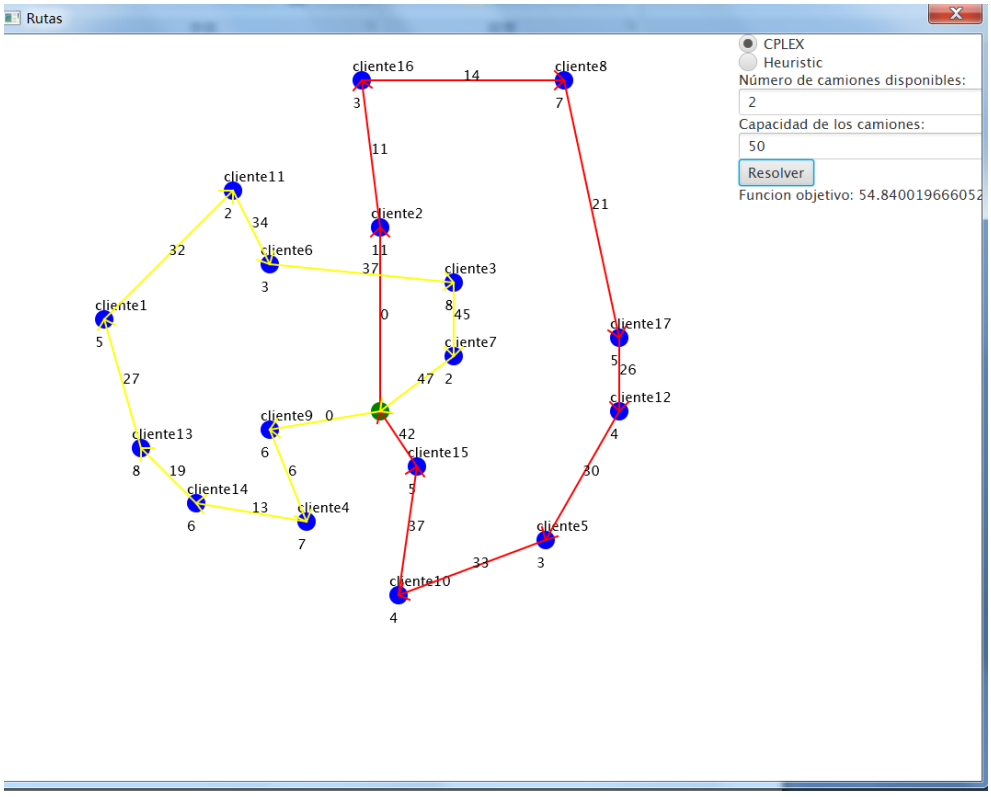

Figure 3: Estado del problema con 17 clientes.

y la capacidad de los camiones para configurar el problema, las unidades de estos parámetros son tienen relevancia.

Para la realización de las pruebas se ha fijado el número de camiones, porque se considera que cada camión por media debería ser capaz de repartir cerca de 10 clientes, así que no tendría mucho sentido hacer pruebas variando el número de camión. Se puede observar en la tabla que se ha variado ligeramente la capacidad de los camiones, eso es poruqe si aumentamos el número de clientes, la demanda total aumenta, si fijamos el número de camiones es posible que el problema se vuelva no factible.

A continuación se ha analizada también la diferencia entre el valor óptimo de la función objetivo obtenido con el método de CPLEX y el método heurístico.

FO: CPLEX	FO: Método heurístico(ms)	Diferencia
28.96	28.96	0
34.7	37.33	2.63
36.75	40.54	3.79
39.32	43.5	4.18
42	48.9	6.9
44.85	51.82	6.97
46.8	58.64	11.84
48.87	64.9	16.03
50.61	67.46	16.85
52.98	72.96	19.98
54.84	75.42	20.58

Figure 4: Estado del problema con 17 clientes.

Podemos observar claramente que es muy difícil que el método heurístico logre obtener el valor óptimo incluso con un tiempo de ejecución mucho menor. Además, podemos observar que la diferencia entre el valor óptimo obtenido es cada vez más lejano, parece ser que mientras más se incremente el número de clientes más impreciso es la obtención del valor óptimo.

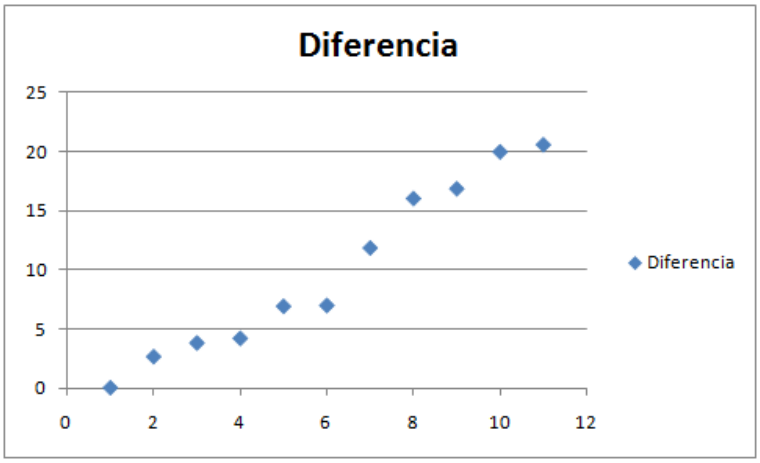

Figure 5: Table de tiempos de ejecución.

7 Implementación del sistema

7.1 Herramientas y lenguajes utilizados

El lenguaje de programación que se ha utilizado para desarrollar el programa es Java. Al ser un proyecto con cierta complejidad, es más adecuado un lenguaje orientado a objetos. Se ha escogido Java principalmente por mi dominio sobre este lenguaje, por su facilidad de uso, y sobretodo, por la gran cantidad de librerías útiles que dispone.

El entorno de desarrollo utilizado es *Eclipse*, principalmente porque estoy familiarizado con el entorno y por la facilidad de incorporar aplicaciones complementarias como las que se mencionará a continuación.

7.1.1 JavaFX

JavaFX es una plataforma que fue creada por la empresa *Oracle*, es usada principalmente para la creación de aplicaciones de escritorio, permite la implementación de interfaces y soporta gran variedad de sistemas operativos. Estas son algunas de las ventajas de JavaFX:

- Permite a los creadores de contenido arrastrando y soltando componentes de interfaces hacie el escritorio.
- Soporta multiplataforma
- Utiliza el mismo lenguaje para la web, para el escritorio y para la telefonía móvil.
- Es open source.

En *JavaFX* la interfaz es manejada por un archivo ".fxml" que usa un lenguaje *xml*. Esto hace que sea mucho más simple la configuración y los cambios de los parámetros que definen las características de los componentes.

Para diseñar la interfaz gráfica se puede hacer de dos maneras: mediante código importando las clases de los componentes, y utilizando el *Scene builder* que permite arrastrar los componentes a las ventanas de la interfaz. Este segundo método es mucho más fácil de utilizar y permite manejar libremente los componentes con gran agilidad

Para manejar las interfaces se crean las clases llamadas controladores. A cada ventana de la interfície gráfica se asigna un controlador. Dentro de la clase controlador se definen los métodos y los *Handlers* de los eventos que se producen cuando el usuario interacciona con la interfaz. Otra de las características principales es que desde los controladores también se permite acceder a los campos que definen las propiedades de los componentes.

7.1.2 SQL

En el proyecto se necesita utilizar una base de datos para guardar la información de los clientes, esta información consiste en el nombre, la localización en coordenadas, y el último pedido que ha realizado. Todo esto con la finalidad de simular también el proceso de pedido de las tiendas. Para manejar las bases de datos se ha utilizado SQL.

SQL(*Structured Query Language*) es un lenguaje utilizado para el manejo de datos en un sistema de base de datos relacional.

Estas son algunas de las ventajas que conlleva usar SQL:

- No tenemos que tener en cuenta cómo estarán constituidos los ficheros de los bases de datos para realizar operaciones.
- Atomicidad: esto quiere decir que cualquier operación en la base de datos está garantizada que si surge algún tipo de problema, la información no se completará, así que o se realiza completamente o no se realiza nada.
- Estándares bien definidos: es decir, todas las operaciones se escriben con la misma sintaxis bajo los estándares de SQL.
- Sencillez en la estructura: son fáciles de comprender.

En el proyecto se ha creado una base de datos con *SQLite Maneger*, un sistema de gestión de bases de datos que permite incorporarse a los navegadores. Se ha utilizado este sistema principalmente por la facilidad de instalación: simplemente se ha añadido como extensión para el navegador. Permite realizar cualquier operación con SQL y descargar un fichero

de base de datos

Una vez descargado el fichero se ha incorporado en la carpeta del proyecto del sistema, que posteriormente se manejará con llamadas SQL desde código Java.

7.1.3 AMPL

AMPL (*A Mathematical Programming language*) es un lenguaje de programación algebraica para escribir y solucionar problemas de gran complejidad matemática de gran escala. Una de las características principales es la semejanza de su sintaxis a la notación matemática de los problemas de optimización.

En el proyecto se ha utilizado AMPL para la resolución del problema en la parte de programación lineal entera. El programa creado en AMPL tiene una sintaxis simple y entendedora.
El uso de AMPL consiste en crear un fichero ".mod" que define el modelo del problema a resolver, y un fichero ".dat" que asigna valores a los parámetros del problema que se necesita resolver.

Una vez creado los ficheros de modelos y datos, se llama a los *solvers* para resolver los problemas representados en AMPL. El *solver* utilizado en el proyecto es CPLEX, un optimizador capaz de resolver problemas de programación lineal y problemas de programación lineal entera.

7.2 Estructura del proyecto

Para empezar, presentaré las ventanas que contiene la aplicación:

- Menú principal: será la primera ventana que verá el usuario. En este menú aparecen dos funcionalidades: cargar cliente y dar de alta el cliente.

- Alta de cliente: ventana que contiene un pequeño formulario que sirve para rellenar la información de los clientes. Los campos a rellenar son: nombre del cliente (nombre de la empresa de la tienda distribuidora) y las coordenadas x e y. Efectivamente no pueden existir dos clientes con el mismo nombre o dos clientes con exactamente las mismas coordenadas.

- Cargar clientes: aparece una lista de los clientes que hay en la base de datos y muestra la cantidad de productos en cajas que ha pedido cada cliente.

- Modificar cliente: aparece cuando se selecciona un cliente en la ventana anterior y se aprieta a "Modificar Información". Consiste en una

ventana igual que la de Alta de cliente, sólo que con la información del cliente rellenada, desde aquí se permite al usuario modificar la información de los clientes. Cuando se da al botón de "Confirmar" se actualiza la base de datos, en caso contrario no se guarda los cambios.

- Modificar pedido: cuando se hace doble clic sobre un cliente aparece esta ventana. Está constituida por una lista de productos y una lista de cantidades para elegir la cantidad de cajas que se quiere pedir de cierto producto. Obviamente, se está haciendo una clasificación muy general de los productos debido a la inmensa cantidad de productos existentes en la realidad que en este producto no es abarcable.

- Rutas: en esta ventana se puede visualizar el resultado de las rutas obtenidas utilizando el método de resolución correspondiente. El resultado se muestra en forma de mapa de grafo, donde el nodo verde corresponde al almacén y los nodos azules representan las tiendas de distribución

La base de datos consiste en una tabla "cliente_table" que contiene las columnas: nombre, coordenadaX, coordenadaY y ultimoPedido. Donde se guarda respectivamente: el nombre, las coordenadas y el últimoPedido realizado por un cliente.

A continuación se presentan los ficheros que componen el proyecto:

- Main.java: crea la aplicación y los componentes necesarios para mostrar el menú principal.

- MainMenuController.java: controlador de la ventana principal, contiene la implementación de los *handlers* para abrir las ventanas de "Alta cliente" y "Cargar clientes".

- AltaClienteController.java: controlador de la ventana "Alta cliente", se encarga de recoger los datos introducidos por el usuario, conectarse a la base de datos, verificar si es posible dar de alta a un cliente con la información introducida, y en caso de éxito, actualizar la base de datos.

- CargarClienteClienteController.java: controlador de la ventana "Cargar clientes", contiene la implementación de los *handlers* para abrir las ventanas de "Modificar cliente" y "Modificar pedido". También se conecta a la base de datos para cargar los clientes dados de alta.

- ModificarClienteController.java: controlador de la ventana "Modificar clientes", se conecta a la base de datos para comprobar si las modificaciones se pueden realizar, en caso de éxito modifica la base de datos.

- ProductosController.java: controlador de la ventana "Modificar pe-

dido", se conecta a la base de datos para modificar el último pedido de los clientes.
- CPLEXSolver.java: clase que se encarga de la resolución del problema con el método de la programación lineal entera. Contiene la creación de los ficheros ".mod" y ".dat". Utiliza una API de *AMPL* para realizar la optimización.

- HeuristicSolver.java: clase que se encarga de la resolución del problema con el método heurístico *Simulated annealing*. Dentro de la clase se distingue la parte de la fase constructiva y la fase iterativa.

- MapaControler.java: clase que se encarga de dibujar todos los componentes del mapa de grafo. Se distingue la parte de dibujo de los nodos y la parte de las aristas.
- ClienteHandler.java: es una clase que contiene métodos para realizar operaciones (*querys*) a la base de datos.

- Cliente.java: es la clase que representa el modelo del cliente. Almacena la información de un cliente como el nombre, las coordenadas y el último pedido realizado. La clase tiene como atributos las columnas de las tablas de la base de datos.

- SqliteConnection.java: simplemente contiene unas instrucciones para conectar con la base de datos.

- MainMenu.fxml: es la interfaz de la ventana principal de la aplicación.

- AltaCliente.fxml es la interfaz de la ventana "Alta cliente"

- ModificarCliente.fxml: es la interfaz de la ventana "Modificar cliente"

- CargarCliente.fxml: es la interfaz de la ventana "Cargar clientes"

- Productos.fxml: es la interfaz de la ventana "Modificar pedidos"

- application.css: es un fichero que sirve para las características de diseño de la interfície, por ejemplo el color de fondo, la mida de los componentes, etc. Es una peculiaridad del *JavaFX* que permite al creador de contenidos personalizar el diseño de la interfaz de la aplicación fácilmente.
- sample.mod: es el fichero modelo del problema de programación lineal generado por la clase "CPLEXSolver.java"

- sample.dat: es el fichero de datos del problema de programación lineal generado por la clase "CPLEXSolver.java"

- clientes.sqlite: es el fichero de base de datos.

8 Posibles mejoras

Debido a los recursos limitados de un trabajo de fin de carrera, este proyecto podría tener varias ampliaciones, estas ampliaciones lo haría un proyecto más completo e interesante, a la vez, si se pretende comercializar el producto sería imprescindible conseguir un grado más alto de complejidad para poder competir con otros productos del mercado.

Una de las posibles mejoras mencionadas anteriormente es integrar el mapa de grafos con un mapa real de nuestra ciudad y guardar direcciones reales en la información, para poner el producto en comercialización es totalmente necesario, ya que al usuario no le interesará un mapa ficticio, ni se molestará a abstraer la realidad en forma de coordenadas para adaptarse a nuestro producto.

Otras de las consideraciones que debería tener el proyecto en cuenta es el concepto de tiempo, ya que en la vida real los repartos se realizan durante todo el día, y un camión realiza más de un itinerario al día. Para una futura implementación, estaría bien permitir a las tiendas introducir el tiempo o el turno de preferencia de reparto (mañanas o tardes).

Una de las variantes que podría tener el problema en la vida real respecto el proyecto, es la variación del número de camiones según la necesidad de las demandas, y que la carga máxima de los camiones sean diferentes. El sistema debería poder permitir obtener el número mínimo de camiones necesarios para obtener el mejor beneficio, y tambien determinar la mida o capacidad máxima de los camiones necesarios.

Una funcionalidad que se podría incorporar al proyecto sería permitir al usuario añadir sus propias restricciones, como por ejemplo repartir a ciertos clientes con preferencia para contentarlos.

En la definición de coste del problema se podría tener en cuenta algunos factores que existen en la vida real, como por ejemplo el sueldo diario de un conductor, el clima (si llueve por la mañana a lo mejor sale más a cuenta hacer el reparto por la tarde), etc.

Otra de las mejoras posibles sería mejorar la interfaz gráfica del programa y hacerla más atractiva. Así como incorporar más funcionalidades para facilitar la tarea del usuario, como permitir interacciones en el mapa una vez se visualiza el resultado.

Por último, la implementación del método heurístico siempre tiene margen de mejora por definición de heurística. Las estrategias de asignación de rutas y los parámetros se pueden ajustar con más experimentación futura para conseguir acercarse más a la solución óptima.

9 Conclusión

Analizando las diferencias entre los dos métodos de resolución de problema he podido contemplar la gran diferencia que existe entre ellos. Esto hace que tengamos que elegir entre precisión (programación lineal entera) y tiempo(métodos heurísticos).

Siendo estos las herramientas más potentes para la resolución de estos problemas, podemos utilizarlas alternativamente dependiendo del contexto o nuestro interés. Por ejemplo si la solución óptima que encontramos con los métodos heurísticos difieren poco con el óptimo, podemos sacrificar esa diferencia de ingresos y conformarnos con la solución encontrada. Por el otro lado, si la solución óptima encontrada es mucho pero que el óptimo, quizás vale la pena esperar a que el optimizador de programación lineal entera nos dé la mejor solución.

Al realizar este trabajo, he profundizado mis conocimientos adquiridos durante mi carrera. Entre ello incluye el desarrollo de una aplicación, el diseño de la interfaz, la implementación de algoritmos avanzados de computación, aplicación de bases de datos, etc.

Con el aprendizaje en GEP me ha permitido conocer mejor cómo desarrollar una tesis, sobretodo me ha inculcado valores de planificación para realizar un buen trabajo.

Con la resolución de un problema de optimización sobre planificación de rutas me ha ayudado a tener un conocimiento más amplio sobre la investigación operativa. También ha despertado en mí un fuerte interés realizar proyectos relacionados con el sector en el futuro.

10 Anexo

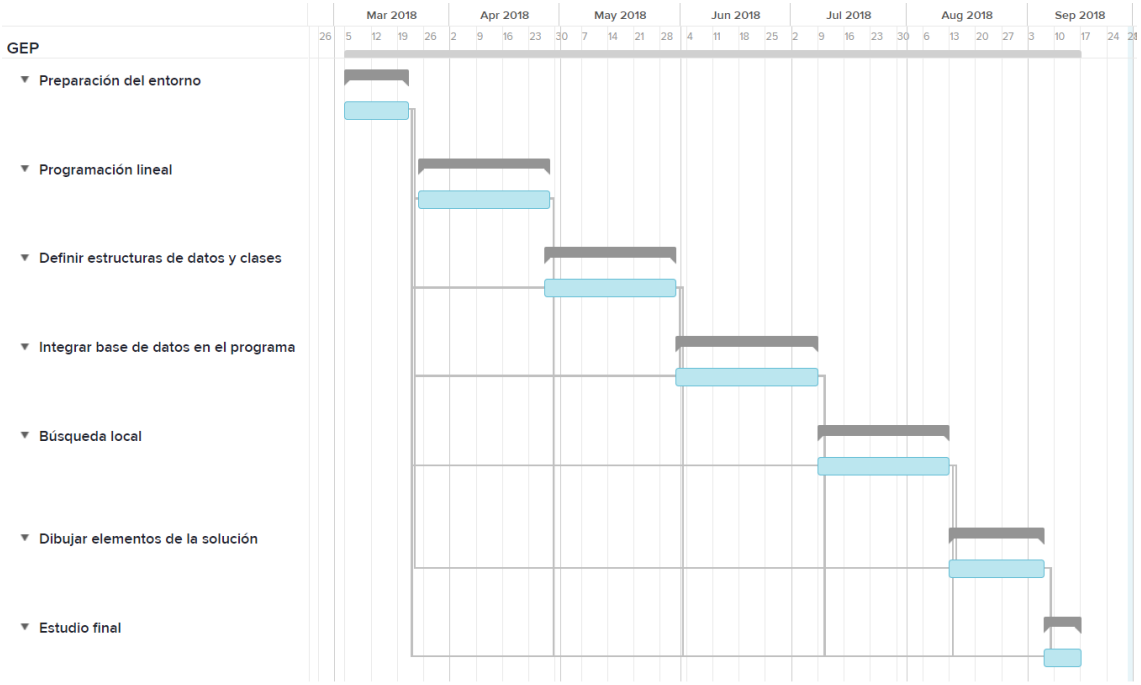

Figure 6: Diagrama de Gantt de las tareas. Generada en: **https://www.teamgantt.com**

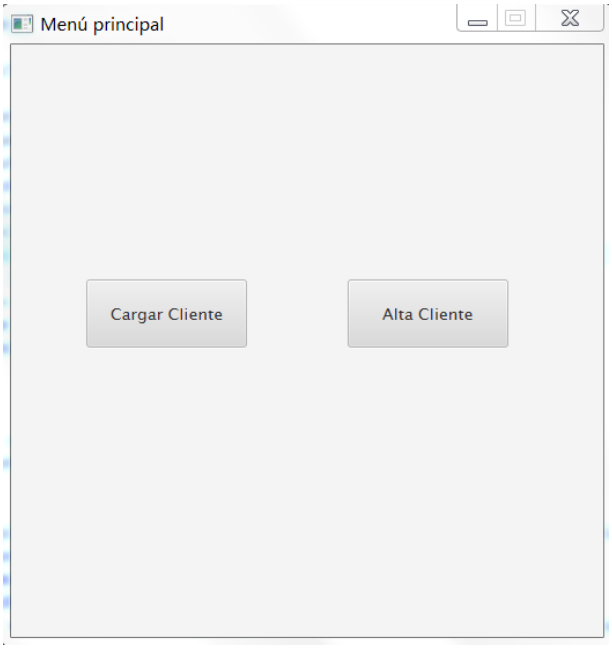

Figure 7: Menú: ventana principal

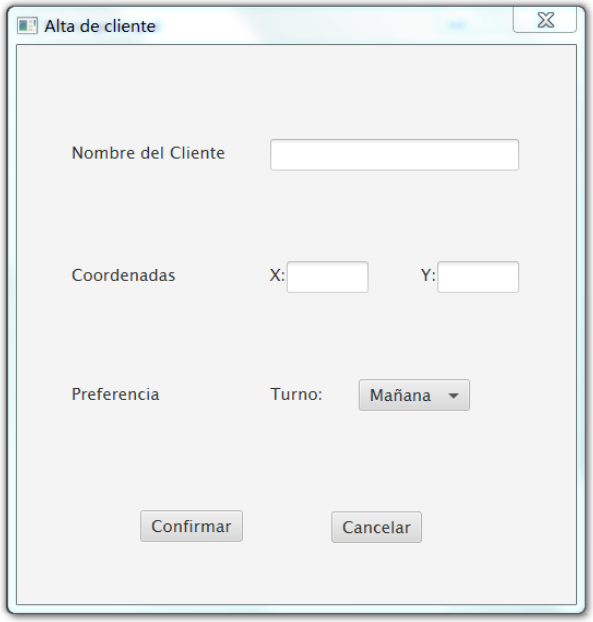

Figure 8: Menú: alta de cliente

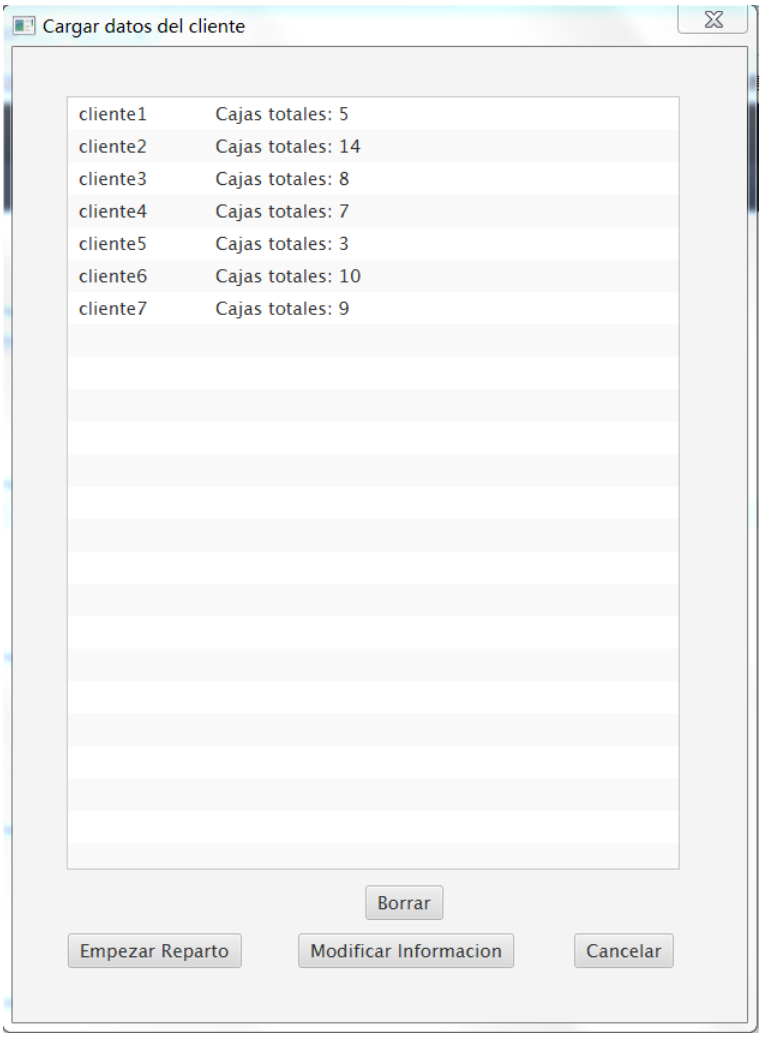

Figure 9: Menú: cargar clientes

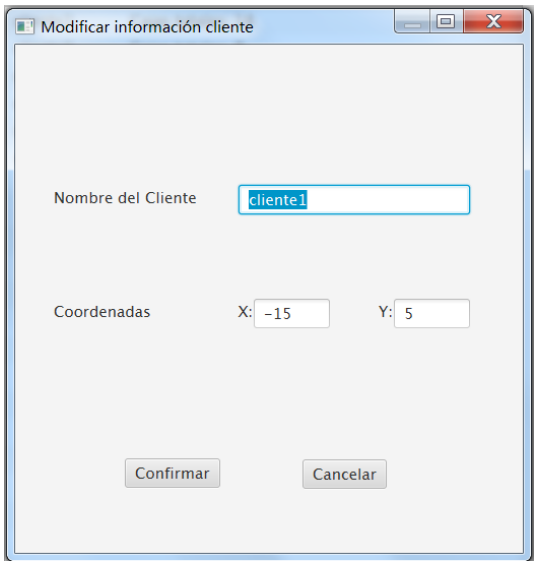

Figure 10: Menú: modificar información clientes

Figure 11: Menú: hacer pedido

```java
//Parametros del simulated annealing
int temperatura = 10;
int temperatura2 = 10;
double coolingRate = (double) 0.003;
double coolingRate2 = (double) 0.003;

class Ruta
{
    public double distancia;
    public Pair<Integer,Integer> posicion;
};
```

Figure 12: Declaración de parámetros del *Simulated annealing*

```java
private ArrayList<ArrayList<Integer> > faseConstructiva() { //Algoritmo de clarkeAndWright
    construirMatrizAhorros();
    ArrayList<ArrayList<Integer> > resultado = asignarRutas();
    return resultado;
}

private void construirMatrizAhorros() {
    Comparator<Ruta> comparador = new Comparator<Ruta>() {
        @Override
        public int compare(Ruta r1, Ruta r2) {
            return Double.compare(r2.distancia, r1.distancia);
        }
    };

    double[][] matrizAhorros = new double[listaClientes.size()][listaClientes.size()];
    matrizAhorrosOrdenada = new PriorityQueue<> (comparador);
    treeSet = new TreeSet<Ruta>();

    for (int i = 0; i < listaClientes.size(); ++i) {
        for (int j = 0; j < i; ++j) { ////Cuidado en calcular el coste de esta funcion, tendria que ser menor que cuadrado
            int ix = listaClientes.get(i).getCoordenadaX();
            int iy = listaClientes.get(i).getCoordenadaY();
            int jx = listaClientes.get(j).getCoordenadaX();
            int jy = listaClientes.get(j).getCoordenadaY();
            matrizAhorros[i][j] = matrizAhorros[j][i] =
                    distancia(ix,iy,0,0) + distancia(0,0,jx,jy) - distancia(ix,iy,jx,jy);
            Ruta ruta = new Ruta();
            ruta.distancia = distancia(ix,iy,0,0) + distancia(0,0,jx,jy) - distancia(ix,iy,jx,jy);
            ruta.posicion = new Pair<Integer,Integer> (i+1,j+1);
            matrizAhorrosOrdenada.add(ruta);
        }
    }
```

Figure 13: Implementación de la fase constructiva

```
private ArrayList<ArrayList<Integer> > asignarRutas() {
    ArrayList<Integer> anterior = new ArrayList<Integer>(Collections.nCopies(listaClientes.size(), 0));
    ArrayList<Integer> posterior = new ArrayList<Integer>(Collections.nCopies(listaClientes.size(), 0));
    while (!matrizAhorrosOrdenada.isEmpty()) {
        Ruta ruta = matrizAhorrosOrdenada.remove();
        Pair<Integer,Integer> pair = ruta.posicion;
        if (anterior.get(pair.getValue()-1) == 0) {
            if (posterior.get(pair.getKey()-1) == 0) {
                int sumaDemandaPost = calcularDemandaRuta(posterior,pair.getValue());
                int sumaDemandaAnt = calcularDemandaRuta(anterior,pair.getKey());
                if (sumaDemandaPost + sumaDemandaAnt <= capacidadCamion) {
                    posterior.set(pair.getKey()-1,pair.getValue());
                    anterior.set(pair.getValue()-1, pair.getKey());
                }
            }
        }
    }
    ArrayList<ArrayList<Integer>> resultado = new ArrayList<ArrayList<Integer> >();
    //Por como esta hecho al menos deberia haber un nodo con un anterior 0 y un posterior 0
    for (int i = 0; i < anterior.size(); ++i) {
        if (anterior.get(i) == 0) {
            ArrayList<Integer> arrayList = new ArrayList<Integer>();
            int j = i; //J tendra siempre valor indice, no valor nodo
            arrayList.add(0);
            arrayList.add(j+1);
            while (posterior.get(j) != 0) {
                j = posterior.get(j)-1;
                arrayList.add(j+1);
            }
            arrayList.add(0);
            resultado.add(arrayList);
        }
    }

    return resultado;
}
```

Figure 14: Método de asignación de rutas

```
private ArrayList<ArrayList<Integer> > simulatedAnnealing(ArrayList<ArrayList<Integer> > resultado) {
    double energia = calcularFuncionObjetivo(resultado);
    ArrayList<ArrayList<Integer> > mejorResultado = copiarArrayArrayList(resultado);
    boolean intercambioDiferentesRutas = false;
    while (temperatura2 > 1) {

        ArrayList<ArrayList<Integer> > nuevoResultado = copiarArrayArrayList(resultado);
        //Intercambio de clientes entre diferentes rutas
        if (intercambioDiferentesRutas) {
            if (nuevoResultado.size() > 1) {

                int cliente = nuevoResultado.get(0).get(1);
                int posicionIDestino = (int) (nuevoResultado.size() * Math.random());
                while (posicionIDestino == 0) {
                    posicionIDestino = (int) (nuevoResultado.size() * Math.random());
                }
                int posicionJDestino = (int) (nuevoResultado.get(posicionIDestino).size() * Math.random());
                while (posicionJDestino == 0) {
                    posicionJDestino = (int) (nuevoResultado.get(posicionIDestino).size() * Math.random());
                }
                nuevoResultado.get(posicionIDestino).add(posicionJDestino, cliente);

                nuevoResultado.get(0).remove(1);
                if (nuevoResultado.get(0).size() == 2) {
                    nuevoResultado.remove(0);
                }
            }
        }
        intercambioDiferentesRutas = true;
```

Figure 15: Implementación de la fase de exploración

```
if (rutaFactible(nuevoResultado)) {
    System.out.println(nuevoResultado);
    resultado = copiarArrayArrayList(nuevoResultado);
    temperatura = 10;
    //Intercambio de clientes en las mismas rutas
    while (temperatura > 1) {
        // Generar un nuevo resultado
        nuevoResultado = copiarArrayArrayList(resultado);
        int cliente1 = (int) (listaClientes.size() * Math.random()) + 1;
        int cliente2 = (int) (listaClientes.size() *  Math.random()) + 1;
        while (cliente1  == cliente2) {
            cliente2 = (int) (listaClientes.size() *  Math.random()) + 1;
        }
        boolean cliente1Encontrado = false, cliente2Encontrado = false;
        for (int i = 0; i < nuevoResultado.size() && !(cliente1Encontrado && cliente2Encontrado); ++i) {
            for (int j = 0; j < nuevoResultado.get(i).size() && !(cliente1Encontrado && cliente2Encontrado); ++j) {
                if (nuevoResultado.get(i).get(j) == cliente1) {
                    nuevoResultado.get(i).set(j, cliente2);
                    cliente1Encontrado = true;
                }
                else if (nuevoResultado.get(i).get(j) == cliente2){
                    nuevoResultado.get(i).set(j, cliente1);
                    cliente2Encontrado = true;
                }
            }
        }
        double nuevaEnergia = calcularFuncionObjetivo(nuevoResultado);
        System.out.println("Nuevo resultado provisional: " + nuevoResultado + " Funcion objetivo: " + nuevaEnergia);
```

Figure 16: Implementación de la fase de exploración

```
                if (rutaFactible(nuevoResultado)) {
                    // Evaluacion del nuevo resultado
                    if ((mejorResultado.size() > numeroCamiones && nuevoResultado.size() < mejorResultado.size())
                            || acceptanceProbability(energia, nuevaEnergia) > Math.random()) {
                        resultado = copiarArrayArrayList(nuevoResultado);
                        energia = calcularFuncionObjetivo(resultado);
                        System.out.println("Nuevo resultado: " + nuevoResultado + " Funcion objetivo: " + nuevaEnergia);

                        if ((mejorResultado.size() > numeroCamiones && nuevoResultado.size() < mejorResultado.size())
                                || energia < calcularFuncionObjetivo(mejorResultado)) {
                            mejorResultado = copiarArrayArrayList(resultado);
                            System.out.println("Nuevo mejor resultado: " + mejorResultado + " Funcion objetivo: " + nuevaEnergia);
                        }
                    }
                    temperatura *= 1-coolingRate;
                }
                else {
                    System.out.println("Ruta no factible");
                }
            }
        }

        temperatura2 *= 1-coolingRate2;
    }
    return mejorResultado;
}
```

Figure 17: Implementación de la fase de exploración

```
private boolean rutaFactible(ArrayList<ArrayList<Integer> > resultado) {
    boolean factible = true;
    for (int i = 0; i < resultado.size() && factible; ++i) {
        int sumaPeso = 0;
        for (int j = 0; j < resultado.get(i).size() && factible; ++j) {
            int cliente = resultado.get(i).get(j);
            if (cliente != 0) {
                sumaPeso += listaClientes.get(cliente-1).getCajasTotales();
                if (sumaPeso > capacidadCamion) {
                    factible = false;
                }
            }
        }
    }
    return factible;
}
```

Figure 18: Método de comprobación de satisfactibilidad de rutas

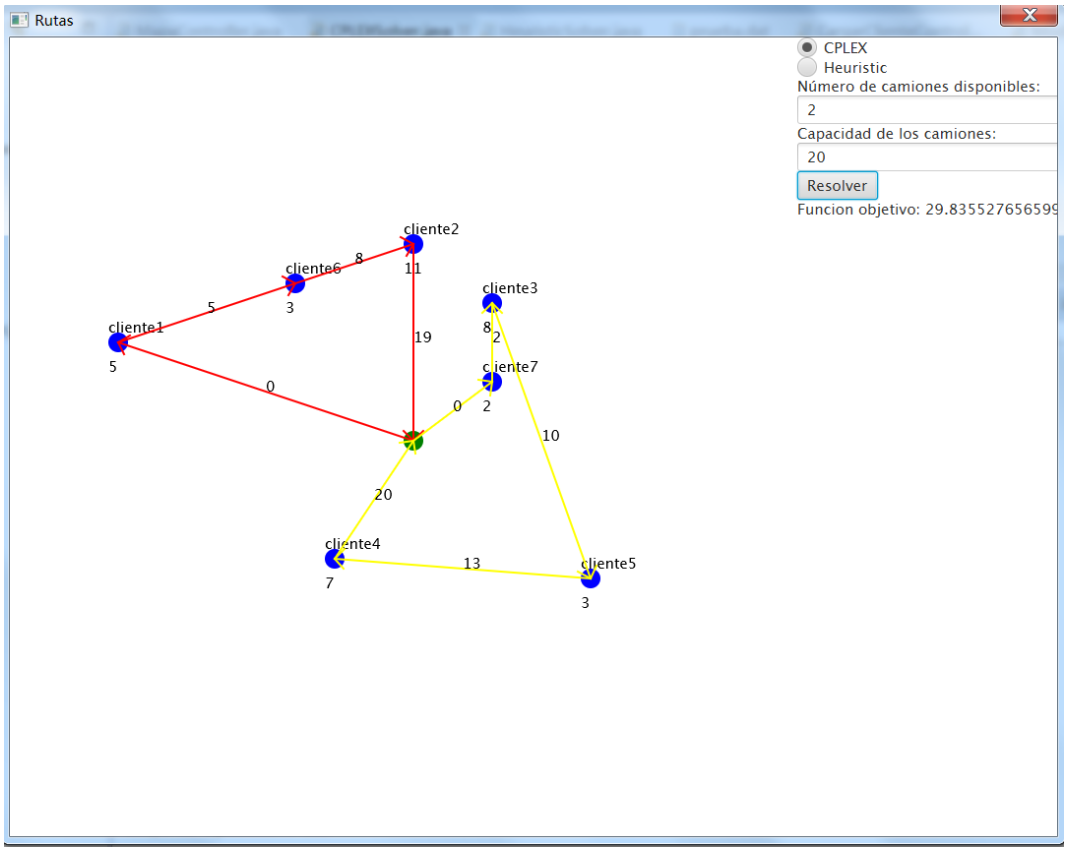

Figure 19: Menú: mapa de grafos

```
private void dibujarAlmacenClientes() {
    gc.setFill(Color.GREEN);
    gc.fillOval(ALMACEN_X, ALMACEN_Y, ANCHURA_NODO, ALTURA_NODO);
    for (int i = 0; i < listaClientes.size(); ++i) {
        Pair<Integer,Integer> pair = posicionRelativa(listaClientes.get(i).getCoordenadaX(), listaClientes.get(i).getCoordenadaY());
        gc.setFill(Color.BLUE);
        gc.fillOval(pair.getKey(),pair.getValue(), ANCHURA_NODO, ALTURA_NODO);

        gc.setFill(Color.BLACK);
        gc.fillText(listaClientes.get(i).getNombre(), pair.getKey(), pair.getValue());
        gc.fillText(String.valueOf(listaClientes.get(i).getCajasTotales()), pair.getKey(), pair.getValue() + ALTURA_NODO * 2);
    }
}
```

Figure 20: Método de dibujo de clientes.

```
private void dibujarAristas() {
    gc.setLineWidth(2);
    for(int i = 0; i < resultado.size(); ++i) {
        int peso = 0;
        for (int j = 0; j < resultado.get(i).size(); ++j) {
            if (j != 0) {
                int nodoActual = resultado.get(i).get(j);
                int nodoAnterior = resultado.get(i).get(j-1);
                colorRuta(i);
                if (nodoActual != 0) {
                    int x = listaClientes.get(nodoActual - 1).getCoordenadaX();
                    int y = listaClientes.get(nodoActual - 1).getCoordenadaY();
                    Pair<Integer,Integer> pair = posicionRelativa(x,y);
                    if (nodoAnterior == 0) {
                        dibujarFlecha(ALMACEN_X, ALMACEN_Y, pair.getKey(), pair.getValue(), peso);
                    }
                    else {
                        int xant = listaClientes.get(nodoAnterior - 1).getCoordenadaX();
                        int yant = listaClientes.get(nodoAnterior - 1).getCoordenadaY();
                        Pair<Integer,Integer> pairAnt = posicionRelativa(xant,yant);
                        dibujarFlecha(pairAnt.getKey(), pairAnt.getValue(), pair.getKey(), pair.getValue(), peso);
                    }
                }
                else { //Asumimos que no existe una ruta que contenga solamente [0,0]
                    int xant = listaClientes.get(nodoAnterior - 1).getCoordenadaX();
                    int yant = listaClientes.get(nodoAnterior - 1).getCoordenadaY();
                    Pair<Integer,Integer> pairAnt = posicionRelativa(xant,yant);
                    dibujarFlecha(pairAnt.getKey(), pairAnt.getValue(), ALMACEN_X, ALMACEN_Y, peso);
                }
                peso = pesosCamion.get(resultado.get(i).get(j));
            }
        }
    }
}
```

Figure 21: Método de dibujo de aristas.

```java
private Pair<Integer,Integer> posicionRelativa(int x, int y) { //Posicion de los clientes respecto almacen
    int resx = x * ANCHURA_NODO + ALMACEN_X;
    int resy = -y * ALTURA_NODO + ALMACEN_Y;
    return new Pair<Integer,Integer> (resx,resy);
}

private void dibujarFlecha(int origenx, int origeny, int destinox, int destinoy, int peso) {
    int orx = origenx + OFFSET, ory = origeny + OFFSET, destx = destinox + OFFSET, desty = destinoy + OFFSET;
    gc.strokeLine(orx, ory, destx, desty);

    //Escribir el peso restanto
    gc.setFill(Color.BLACK);
    gc.fillText(String.valueOf(peso), (double) (destx + orx) / 2.0, (double) (desty + ory) / 2.0);

    //Calcular un punto sobre la recta. Nuestro vectorunitario va alrevés
    double vectorUnitariox = (orx - destx) / Math.sqrt((destx - orx) * (destx - orx) + (desty - ory) * (desty - ory));
    double vectorUnitarioy = (ory - desty) / Math.sqrt((destx - orx) * (destx - orx) + (desty - ory) * (desty - ory));
    double puntox = orx, puntoy = ory;
    puntox = destx + vectorUnitariox * OFFSET;
    puntoy = desty + vectorUnitarioy * OFFSET;

    //Calcular los dos puntos que forman el arco de la flecha
    double puntoFinal1x = puntox + (-vectorUnitarioy * OFFSET); //Vector unitario perpendicular al anterior
    double puntoFinal1y = puntoy + (vectorUnitariox * OFFSET);
    double puntoFinal2x = puntox - (-vectorUnitarioy * OFFSET); //Vector unitario perpendicular al anterior
    double puntoFinal2y = puntoy - (vectorUnitariox * OFFSET);

    /*
    System.out.println("Puntofinal1 :" + puntoFinal1x + "Puntofinal1 :" + puntoFinal1y);
    System.out.println("Puntofinal2 :" + puntoFinal2x + "Puntofinal2 :" + puntoFinal2y);
     */

    gc.strokeLine((int) puntoFinal1x, (int) puntoFinal1y, destx, desty);
    gc.strokeLine((int) puntoFinal2x, (int) puntoFinal2y, destx, desty);
}
```

Figure 22: Método de dibujo de los arcos de las flechas.

11 Referencias

[1]: Libro sobre *VRP*:
Vehicle Routing Problems, Methods, and Applications. *Paolo Toth y Daniele Vigo*. Editorial 2014

[2]: A Guide to the Theory of NP-Completeness. *Garey & Johnson*. W. H. Freeman & Co. New York, NY, USA 1979

[3]: Apuntes de búsqueda local:
LSI-FIB-UPC: BH3-BúsquedaLocal.pdf

[4]: Explicación Clark and wright:
`https://www.youtube.com/watch?v=ZlDWHSHYX6Q`

[5]: Manual de referencias JavaFX:
`https://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm`

[6]: Salario ingeniero software:
`https://www.indeed.es/salaries/Desarrollador/a-de-software-Salaries?period=hourly`

[7]: Manual de referencias AMPL:
`https://ampl.com/`

[8]: Generación del *Gantt Chart*:
`https://prod.teamgantt.com`

[9]: Escritor *Online* de Latex:
`https://www.overleaf.com`

[10]: Tutorial de *Simulated annealing*:
`http://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginne`

[11]: Tutorial de *JavaFX*:
`http://www.java2s.com/Tutorials/Java/JavaFX/`

[12]: Características de *JavaFX*:
`https://williamrsl.wordpress.com/javafx/`

[13]: Características de SQL:
`https://medium.com/@marlonmanzo/sql-vs-nosql-ventajas-y-desventajas-849ccc9db3d4`